Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 174 (2007) 1–4
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

# Preface

As the quest for *correct* high performance software and hardware systems continues, thread level modeling and verification have attained a position of central importance. At the hardware level, chip level multiprocessors offer a path towards higher performance per unit of energy consumption. At higher levels of hardware and software, the use of threading renders applications modular by separating concerns, and offers a natural path towards the exploitation of multicores. Multiple paradigms often need to be employed in one setting: for example, the use of threading, objects, and linguistic mechanisms. As the debugging costs of computing systems skyrocket, a variety of formal and semi-formal verification methods – including model checking, static checking, assertion proving using decision procedures, and systematic random testing – require domain-specific development and advancement.

Out of the many workshops and symposia being organized on the above topics, the Thread Verification (TV) workshop held as part of The Federated Logic Conference (FLoC) of 2006 featured many invited and contributed papers, and hosted many informal as well as formal discussions among its 36 registrants. This special issue of ENTCS hopes to archive some of the proceedings of TV06. We include eight papers that received uniformly high reviews. These papers have been significantly expanded and revised, and are presented here. We thank the reviewers who conducted very thorough reviewing in what was one of 34 workshops during FLoC! The papers included here fall into three classes: debugging and testing methods, shared memory consistency models, and formal modeling and verification. Under debugging and testing methods, we include four papers, under memory consistency models - two, and under formal modeling and verification - two.

Cook's paper "Thread Verification – an experience report" narrates the author's experience across several thread programming projects. The author touches on the prevalence of thread bugs, and the efficacy of finite state modeling and analysis in detecting such bugs. This paper summarizes several real-world experiences, and also includes thread verification benchmarks that could be, after documentation and elaboration, widely used.

Mühlenfeld and Wotawa's paper "Fault Detection in Multi-Threaded C++ Server Applications" is a case study in the use, and subsequent adaptation for improvement, of Helgrind (a widely used dynamic race detection tool) for debug-


doi:10.1016/j.entcs.2007.04.003

ging large C++ server applications. Techniques to reduce false warnings are the main contribution of the paper, as are results from a 500 kLOC program.

Copty and Ur's paper "Toward Automatic Concurrent Debugging via Minimal Program Mutant Generation with AspectJ" addresses how to instrument a multithreaded program so as to modify its schedule, increasing the likelihood of traversing erroneous paths. When an error is triggered, the instrumentations help pinpoint the source of the error. Details of how AspectJ has been modified in the process of creating their tool framework are discussed.

Regehr and Cooprider's paper "Interrupt Verification via Thread Verification" compares and contrasts threads and interrupts from the point of view of verifying the absence of race conditions. With the prevalence of embedded devices, the importance of verifying interrupt-based systems is bound to escalate. Source to source transformations that transform interrupt-based code into equivalent (for verification) thread-based code are presented.

Ziarek, Schatz, and Jagannathan's paper "Modular Checkpointing for Atomicity" presents a modular checkpointing scheme to ensure atomicity. Ascribing a meaningful re-execution semantics for multithreaded programs is linguistically supported using the notion of *Stabilizers*. Experiments performed in the context of concurrent ML (CML) programming on examples such as web servers and windowing tool kits reveal that stabilizers introduce very low overheads. The construction of open nested transactions using stabilizers is also discussed.

Jacobs, Smans, Piessens, and Schulte's paper "A Simple Sequential Reasoning Approach for Sound Modular Verification of Mainstream Multithreaded Programs" presents the difficulties of reasoning about object-oriented threaded programs owing to the non-local nature of object aliasing, data races, and deadlocks. In their work, Java or `C#` programs are annotated to facilitate the verification of its contracts, including assertions, and the absence of races and deadlocks – all amounting to show compliance with the programming model. Their ideas are built into the `Spec#` verification environment, and have been applied to case studies.

Maessen and Arvind's paper "Store Atomicity for Transactional Memory" presents a method to reason about memory orderings and store atomicity in the presence of transactions. They present a procedure for enumerating the behaviors of a transactional program. They also show that more realistic models of transactional execution require speculation. They present conditions under which speculation must be rolled back.

Last but not least, Higham, Jackson, and Kawash present a detailed analysis of the published Itanium® processor family shared memory consistency model. They show that their attempts to define this memory model in their framework (previously used with success for defining other industrial memory models) does not meet with success, in that they are unable to match the Itanium specification exactly: either their model ends up being weaker, or ends up being stronger than the official Itanium memory model. They detail why these variations occur, and provide many other comparative remarks as well as formal proofs.

The TV workshop also featured several invited talks, as well an open-mike session

where a panel consisting of the invited speakers and some members of the audience debated on various issues connected with threading. Many interesting remarks were repeatedly made, and just a samplng of them are noted here to give the reader an inkling of some common beliefs:

**Transactional Memories:** Transactional memories (TMs) are seriously being investigated. Two of our invited speakers, Maurice Herlihy (Brown) and Nir Shavit (Sun Microsystems Inc.), described TMs in detail, and discussed their pros and cons. Among their pros are that TMs can alleviate many of the problems associated with locking: conceptual difficulty, lack of modularity, and the ease of introducing deadlocks. Among their cons are that TMs can still introduce livelocks, and they need to co-exist with traditional memory access mechanisms, such as `mallocs`.

It was generally believed that while transactions are very interesting, they are not a panacea.

**Computing With Inconsistent States:** This idea was repeatedly mentioned by Nir Shavit. There is a heavy price to be paid by any system that seeks to ensure strong consistency of state updates. Strong consistency can not only decrease thread level parallelism, but can also introduce more errors, due to the over-use of synchronizations. Success in designing high-performance threaded systems depends, to a large extent, on a clear understanding of the problem, which then enables one to derive efficient solutions. Clearly, formal specification and verification play a very vital role in these areas.

**Theoretical and Methodological Advances:** The need for theoretical and methodological advances were brought out in the invited talks given by Vijay Saraswat (IBM), Wolfram Schulte (Microsoft) and Bart Jacobs (Leuven).

**Ambitious and Diverse Proposals are Essential:** Manufacturers will soon be shipping almost exclusively multi-core microprocessors, given that putting even two cores can reduce overall energy consumption while increasing performance. The academic community should, therefore, embrace these opportunities wholeheartedly and develop a wide range of solutions for making thread programming safe, intuitive, and well-performing. These, and the many challenges (including memory bandwidth requirements and application library development) were emphasized by our invited speaker Paul Petersen (Intel).

**Benchmarks:** A large variety of programming techniques as well as verification benchmarks need to be assembled, documented, and disseminated. These items must serve multiple needs. PThreads verification case studies must be presented and documented in detail; formal characterizations of communication and synchronization libraries must be developed and popularized among designers; more safe concurrent programming patterns need be developed (just to name a few).

**Education:** The need to educate students and designers is being felt acutely. Students must be exposed to concurrency early in their curriculum. It is clear that finite-state model checking is a concrete topic that can readily be introduced as a self-contained topic or integrated into classes such as Operating Systems. Given

that the whole computer science undergraduate education is ripe for a revamp, the right set of curricular material pertaining to threading must be developed, field tested, and integrated. These issues were discussed by Arvind (MIT) who was also a panelist.

We hope you will enjoy this issue and continue to apply your energies to formal specification and verification issues associated with threading. We thank Microsoft which generously supported TV 2006 through a grant that helped us offer student bursaries.

*Ganesh Gopalakrishnan*
*John O'Leary*